

# Dynamic Backward Slicing of Rewriting Logic Computations<sup>★</sup>

M. Alpuente<sup>1</sup>, D. Ballis<sup>2</sup>, J. Espert<sup>1</sup>, and D. Romero<sup>1</sup>

<sup>1</sup> DSIC-ELP, Universidad Politécnica de Valencia  
Camino de Vera s/n, Apdo 22012, 46071 Valencia, Spain

{alpuente,jespert,dromero}@dsic.upv.es

<sup>2</sup> Dipartimento di Matematica e Informatica  
Via delle Scienze 206, 33100 Udine, Italy  
demis.ballis@uniud.it

**Abstract.** Trace slicing is a widely used technique for execution trace analysis that is effectively used in program debugging, analysis and comprehension. In this paper, we present a backward trace slicing technique that can be used for the analysis of Rewriting Logic theories.

Our trace slicing technique allows us to systematically trace back rewrite sequences modulo equational axioms (such as associativity and commutativity) by means of an algorithm that dynamically simplifies the traces by detecting control and data dependencies, and dropping useless data that do not influence the final result. Our methodology is particularly suitable for analyzing complex, textually-large system computations such as those delivered as counter-example traces by Maude model-checkers.

## 1 Introduction

The analysis of execution traces plays a fundamental role in many program manipulation techniques. Trace slicing is a technique for reducing the size of traces by focusing on selected aspects of program execution, which makes it suitable for trace analysis and monitoring [8].

Rewriting Logic (RWL) is a very general *logical* and *semantic framework*, which is particularly suitable for formalizing highly concurrent, complex systems (e.g., biological systems [5,21] and Web systems [2,4]). RWL is efficiently implemented in the high-performance system Maude [10]. Roughly speaking, a *rewriting logic theory* seamlessly combines a *term rewriting system* (TRS) together with an *equational theory* that may include sorts, functions, and algebraic laws (such as commutativity and associativity) so that rewrite steps are applied *modulo* the equations. Within this framework, the system states are typically

---

<sup>★</sup> This work has been partially supported by the EU (FEDER) and the Spanish MEC TIN2010-21062-C02-02 project, by Generalitat Valenciana PROMETEO2011/052, and by the Italian MUR under grant RBIN04M8S8, FIRB project, Internationalization 2004. Daniel Romero is also supported by FPI-MEC grant BES-2008-004860.

represented as elements of an algebraic data type that is specified by the equational theory, while the system computations are modeled via the rewrite rules, which describe transitions between states.

Due to the many important applications of RWL, in recent years, the debugging and optimization of RWL theories have received growing attention [1,19,20]. However, the existing tools provide hardly support for execution trace analysis. The original motivation for our work was to reduce the size of the counterexample traces delivered by Web-TLR, which is a RWL-based model-checking tool for Web applications proposed in [2,4]. As a matter of fact, the analysis (or even the simple inspection) of such traces may be unfeasible because of the size and complexity of the traces under examination. Typical counterexample traces in Web-TLR are 75 Kb long for a model size of 1.5 Kb, that is, the trace is in a ratio of 5.000% w.r.t. the model.

To the best of our knowledge, this paper presents the first trace slicing technique for RWL theories. The basic idea is to take a trace produced by the RWL engine and traverse and analyze it backwards to filter out events that are irrelevant for the rewritten task. The trace slicing technique that we propose is fully general and can be applied to optimizing any RWL-based tool that manipulates rewrite logic traces. Our technique relies on a suitable mechanism of backward tracing that is formalized by means of a procedure that labels the calls (terms) involved in the rewrite steps. This allows us to infer, from a term  $t$  and positions of interest on it, positions of interest of the term that was rewritten to  $t$ . Our labeling procedure extends the technique in [6], which allows descendants and origins to be traced in orthogonal (i.e., left-linear and overlap-free) term rewriting systems in order to deal with rewrite theories that may contain commutativity/associativity axioms, as well as nonleft-linear, collapsing equations and rules. As in dynamic tracing [14,22], our definition of labeling uses a relation on contexts derived from the reduction relation on terms, where the symbols in the left-hand side of a rule propagate to all symbols of its right-hand side. This labeling relation allows us to make precise the *dynamic dependence* of function symbols occurring in the terms of a reduction sequence on symbols in previous terms in that sequence [14].

*Plan of the paper.* Section 2 summarizes some preliminary definitions and notations about term rewriting systems. In Section 3, we recall the essential notions concerning rewriting modulo equational theories. Section 4 describes the main kinds of labeling and tracing in term rewrite systems. In Section 5, we formalize our backward trace slicing technique for elementary rewriting logic theories. Section 6 extends the trace slicing technique of Section 5 by considering extended rewrite theories, i.e., rewrite theories that may include collapsing, nonleft-linear rules, associative/commutative equational axioms, and built-in operators. Section 7 describes a software tool that implements the proposed backward slicing technique and reports on an experimental evaluation of the tool that allows us to assess the practical advantages of the trace slicing technique. In Section 8, we discuss some related work and then we conclude. Proofs of the main technical results can be found in Appendix A.

## 2 Preliminaries

A many-sorted signature  $(\Sigma, S)$  consists of a set of sorts  $S$  and a  $S^* \times S$ -indexed family of sets  $\Sigma = \{\Sigma_{\bar{s} \times s}\}_{(\bar{s}, s) \in S^* \times S}$ , which are sets of *function symbols* (or operators) with a given string of argument sorts and result sort. Given an  $S$ -sorted set  $\mathcal{V} = \{\mathcal{V}_s \mid s \in S\}$  of disjoint sets of variables,  $T_\Sigma(\mathcal{V})_s$  and  $T_{\Sigma_s}$  are the sets of terms and ground terms of sorts  $s$ , respectively. We write  $T_\Sigma(\mathcal{V})$  and  $T_\Sigma$  for the corresponding term algebras. An *equation* is a pair of terms of the form  $s = t$ , with  $s, t \in T_\Sigma(\mathcal{V})_s$ . In order to simplify the presentation, we often disregard sorts when no confusion can arise.

Terms are viewed as labelled trees in the usual way. Positions are represented by sequences of natural numbers denoting an access path in a term. The empty sequence  $\Lambda$  denotes the root position. By  $\text{root}(t)$ , we denote the symbol that occurs at the root position of  $t$ . We let  $\text{Pos}(t)$  denote the set of positions of  $t$ . By notation  $w_1.w_2$ , we denote the concatenation of positions (sequences)  $w_1$  and  $w_2$ . Positions are ordered by the prefix ordering, that is, given the positions  $w_1, w_2$ ,  $w_1 \leq w_2$  if there exists a position  $x$  such that  $w_1.x = w_2$ .  $t|_u$  is the subterm at the position  $u$  of  $t$ .  $t[r]_u$  is the term  $t$  with the subterm rooted at the position  $u$  replaced by  $r$ . A substitution  $\sigma$  is a mapping from variables to terms  $\{x_1/t_1, \dots, x_n/t_n\}$  such that  $x_i\sigma = t_i$  for  $i = 1, \dots, n$  (with  $x_i \neq x_j$  if  $i \neq j$ ), and  $x\sigma = x$  for any other variable  $x$ . By  $\varepsilon$ , we denote the *empty* substitution. Given a substitution  $\sigma$ , the *domain* of  $\sigma$  is the set  $\text{Dom}(\sigma) = \{x \mid x\sigma \neq x\}$ . By  $\text{Var}(t)$  (resp.  $\text{FSymbols}(t)$ ), we denote the set of variables (resp. function symbols) occurring in the term  $t$ .

A *context* is a term  $\gamma \in T_{\Sigma \cup \{\square\}}(\mathcal{V})$  with zero or more holes  $\square^3$ , and  $\square \notin \Sigma$ . We write  $\gamma[\ ]_u$  to denote that there is a hole at position  $u$  of  $\gamma$ . By notation  $\gamma[\ ]$ , we define an arbitrary context (where the number and the positions of the holes are clarified *in situ*), while we write  $\gamma[t_1, \dots, t_n]$  to denote the term obtained by filling the holes appearing in  $\gamma[\ ]$  with terms  $t_1, \dots, t_n$ . By notation  $t^\square$ , we denote the context obtained by applying the substitution  $\sigma = \{x_1/\square, \dots, x_n/\square\}$  to  $t$ , where  $\text{Var}(t) = \{x_1, \dots, x_n\}$  (i.e.,  $t^\square = t\sigma$ ).

A *term rewriting system* (TRS for short) is a pair  $(\Sigma, R)$ , where  $\Sigma$  is a signature and  $R$  is a finite set of reduction (or rewrite) rules of the form  $\lambda \rightarrow \rho$ ,  $\lambda, \rho \in T_\Sigma(\mathcal{V})$ ,  $\lambda \notin \mathcal{V}$  and  $\text{Var}(\rho) \subseteq \text{Var}(\lambda)$ . We often write just  $R$  instead of  $(\Sigma, R)$ . A rewrite step is the application of a rewrite rule to an expression. A term  $s$  *rewrites* to a term  $t$  via  $r \in R$ ,  $s \xrightarrow{r}_R t$  (or  $s \xrightarrow{r, \sigma}_R t$ ), if there exists a position  $q$  in  $s$  such that  $\lambda$  *matches*  $s|_q$  via a substitution  $\sigma$  (in symbols,  $s|_q = \lambda\sigma$ ), and  $t$  is obtained from  $s$  by replacing the subterm  $s|_q = \lambda\sigma$  with the term  $\rho\sigma$ , in symbols  $t = s[\rho\sigma]_q$ . The rule  $\lambda \rightarrow \rho$  (or equation  $\lambda = \rho$ ) is *collapsing* if  $\rho \in \mathcal{V}$ ; it is *left-linear* if no variable occurs in  $\lambda$  more than once. We denote the transitive and reflexive closure of  $\rightarrow$  by  $\rightarrow^*$ .

Let  $r : \lambda \rightarrow \rho$  be a rule. We call the context  $\lambda^\square$  (resp.  $\rho^\square$ ) *redex pattern* (resp. *contractum pattern*) of  $r$ . For example, the context  $f(g(\square, \square), a)$

<sup>3</sup> Actually, when considering types, we assume to have a distinct  $\square_s$  symbol for each sort  $s \in S$ , and by abuse we simply denote  $\square_s$  by  $\square$ .

(resp.  $d(s(\square), \square)$ ) is the redex pattern (resp. contractum pattern) of the rule  $r : f(g(x, y), a) \rightarrow d(s(y), y)$ , where  $a$  is a constant symbol.

### 3 Rewriting Modulo Equational Theories

An *equational theory* is a pair  $(\Sigma, E)$ , where  $\Sigma$  is a signature and  $E = \Delta \cup B$  consists of a set of (oriented) equations  $\Delta$  together with a collection  $B$  of equational axioms (e.g., associativity and commutativity axioms) that are associated with some operator of  $\Sigma$ . The equational theory  $E$  induces a least congruence relation on the term algebra  $T_\Sigma(\mathcal{V})$ , which is usually denoted by  $=_E$ .

A *rewrite theory* is a triple  $\mathcal{R} = (\Sigma, E, R)$ , where  $(\Sigma, E)$  is an equational theory, and  $R$  is a TRS. Examples of rewrite theories can be found in [10].

Rewriting modulo equational theories [19] can be defined by lifting the standard rewrite relation  $\rightarrow_R$  on terms to the  $E$ -congruence classes induced by  $=_E$ . More precisely, the rewrite relation  $\rightarrow_{R/E}$  for rewriting modulo  $E$  is defined as  $=_E \circ \rightarrow_R \circ =_E$ . A computation in  $\mathcal{R}$  using  $\rightarrow_{R \cup \Delta, B}$  is a *rewriting logic deduction*, in which the *equational simplification* with  $\Delta$  (i.e., applying the oriented equations in  $\Delta$  to a term  $t$  until a canonical form  $t \downarrow_E$  is reached where no further equations can be applied) is intermixed with the rewriting computation with the rules of  $R$ , using an *algorithm of matching modulo*<sup>4</sup>  $B$  in both cases. Formally, given a rewrite theory  $\mathcal{R} = (\Sigma, E, R)$ , where  $E = \Delta \cup B$ , a *rewrite step modulo*  $E$  on a term  $s_0$  by means of the rule  $r : \lambda \rightarrow \rho \in R$  (in symbols,  $s_0 \xrightarrow{r}_{R \cup \Delta, B} s_1$ ) can be implemented as follows: (i) apply (modulo  $B$ ) the equations of  $\Delta$  on  $s_0$  to reach a canonical form ( $s_0 \downarrow_E$ ); (ii) rewrite (modulo  $B$ ) ( $s_0 \downarrow_E$ ) to term  $v$  by using  $r \in R$ ; and (iii), apply (modulo  $B$ ) the equations of  $\Delta$  on  $v$  again to reach a canonical form for  $v$ ,  $s_1 = v \downarrow_E$ .

Since the equations of  $\Delta$  are implicitly oriented (from left to right), the equational simplification can be seen as a sequence of (equational) rewrite steps ( $\rightarrow_{\Delta/B}$ ). Therefore, a *rewrite step modulo*  $E$   $s_0 \xrightarrow{r}_{R \cup \Delta, B} s_1$  can be expanded into a sequence of rewrite steps as follows:

$$\underbrace{s_0 \rightarrow_{\Delta/B} \dots \rightarrow_{\Delta/B} s_0 \downarrow_E}_{\text{equational simplification}} \xrightarrow{\text{rewrite step}/B} u \xrightarrow{r} v \xrightarrow{\text{equational simplification}} v \downarrow_E = s_1$$

Given a finite rewrite sequence  $\mathcal{S} = s_0 \rightarrow_{R \cup \Delta, B} s_1 \rightarrow_{R \cup \Delta, B} \dots \rightarrow_{R \cup \Delta, B} s_n$  in the rewrite theory  $\mathcal{R}$ , the *execution trace* of  $\mathcal{S}$  is the rewrite sequence  $\mathcal{T}$  obtained by expanding all the rewrite steps  $s_i \rightarrow_{R \cup \Delta, B} s_{i+1}$  of  $\mathcal{S}$  as is described above.

In this work, a rewrite theory  $\mathcal{R} = (\Sigma, B \cup \Delta, R)$  is called *elementary* if  $\mathcal{R}$  does not contain equational axioms ( $B = \emptyset$ ) and both rules and equations are left-linear and not collapsing.

<sup>4</sup> A subterm of  $t$  matches  $l$  (modulo  $B$ ) via the substitution  $\sigma$  if  $t =_B u$  and  $u|_q = l\sigma$  for a position  $q$  of  $u$ .

## 4 Labeling and Tracing in Term Rewrite Systems

Labeling an object allows us to distinguish it within a collection of identical objects. This is a useful means to keep track of a given object in a dynamic system. In the following, we introduce a rather intuitive example that allows us to illustrate how the labeling and tracing process work.

*Example 1.* Let  $r_1 : f(x) \rightarrow b$ , and  $r_2 : g(b) \rightarrow m(a)$  be two rewrite rules. Let  $g(f(a))$  be an initial term. Then, by applying  $r_1$  and  $r_2$  we get the execution trace  $\mathcal{T} = g(f(a)) \xrightarrow{r_1} g(b) \xrightarrow{r_2} m(a)$ .

In term rewriting, we distinguish three kinds of labeling according to the information recorded by them in an execution trace.

- (i) The Hyland–Wadsworth labeling [15,23] records the creation level of each symbol. Roughly speaking, from an initial (default) creation level, the accomplishment of a rewrite step increases by one the creation level of the affected symbols. For example, consider the execution trace  $\mathcal{T}$  of Example 1 together with an initial level 0 for all symbols. Then,

$$g^0(f^0(a^0)) \xrightarrow{r_1} g^0(b^1) \xrightarrow{r_2} m^2(a^2)$$

- (ii) The Boudol–Khasidashvili labeling [7,16,17] records the history of the term in execution traces. The general idea is to record in the history the applied rule and the symbols of the redex pattern. This information is taken as the label for the head symbol of the contractum pattern. Consider again Example 1. First, the set of rules is labeled as follows:

$$r_{1_{f(x)}} : f(x) \rightarrow r_{1_{f(x)}} \quad r_{2_{g(b)}} : g(b) \rightarrow r_{2_{g(b)}}(a)$$

Then, the labeling of the execution trace  $\mathcal{T}$  is:

$$g(f(a)) \rightarrow g(r_{1_{f(x)}}) \rightarrow r_{2_{g(b)}}(a)$$

Note that the initial term of this sequence is not labeled, i.e., the initial label is the identity.

- (iii) The Lévy labeling [18] records the history of each symbol in the term. Basically, this labeling combines the previous two labelings and attaches the history on every symbol of the contractum pattern. Let us show an example. As before, consider Example 1. The labeled rules are as follows:

$$r_{1_{f(x)\lambda}} : f(x)^\lambda \rightarrow r_{1_{f(x)\lambda}}^\lambda \quad r_{2_{g(b)\lambda}} : g(b)^\lambda \rightarrow r_{2_{g(b)\lambda}}^\lambda (r_{2_{g(b)\lambda}}^1)$$

and the labeled trace of  $\mathcal{T}$  is:

$$g(f(a))^\lambda (g(f(a))^\lambda (g(f(a))^{1.1})) \rightarrow g(f(a))^\lambda (r_{1_{f(x)\lambda}}^1) \rightarrow r_{2_{g(b)\lambda}}^\lambda (r_{2_{g(b)\lambda}}^1)$$

Note that due to the accumulation of labels, Lévy labels soon become neither readable nor legible. Note also that this labeling keeps the maximal information in a rewrite step.

In this work, we rely on Klop labeling [6], which is inspired by Lévy labeling. Roughly speaking, Klop labeling employs Greek letters and concatenation of Greek letters as labels. That is, given a rewrite step  $t \rightarrow s$ , the symbols of  $t$  are decorated by using Greek letters as labels. Then, a new label  $l$  is formed by concatenating the labels of the redex pattern. Finally,  $l$  is attached to every symbol of the contractum pattern of  $s$ . A formal definition of this labeling adapted to deal with rewriting logic theories is given in Section 5.1.

Given a rewrite step  $t \rightarrow s$ , tracing allows one to establish a mapping among symbols of  $t$  and symbols of  $s$ . Each symbol is mapped according to its location. For example, occurrences of symbols in the context of  $t$ , or in the computed substitution, are traced to the same occurrences in  $s$ . On the contrary, the mapping for the symbols in the redex and contractum patterns depend on the kind of tracing we adopt. Namely, in *static* tracing the symbols do not persist through the execution trace. On other hand, in *dynamic* tracing the symbols of the redex pattern are mapped to all symbols of the contractum pattern. Let us illustrate this by means of an example.

*Example 2.* Consider the rewrite step  $g(f(a)) \xrightarrow{r_1} g(b)$  into the trace  $\mathcal{T}$  of Example 1. By considering the static tracing, the symbol  $f$  within the term  $g(f(a))$  does not leave a trace to the term  $g(b)$  since  $f$  belongs to redex pattern of  $r_1$ . Contrarily,  $f$  dynamically traces to  $b$ . Finally, in both cases the symbol  $a$  is discarded without leaving a trace in the rewrite step.

As for the dynamic tracing relation, the symbols can be partitioned into *needed* and *non-needed*. A symbol is called *needed* if it leaves a trace in the considered rewrite sequence. For instance, in the previous example,  $f$  is a *needed* symbol. Instead  $a$ , which belongs to substitution  $\sigma = \{x/a\}$ , is a *non-needed* symbol. Given an execution trace, the set of needed symbols in a term of the trace forms a prefix which is also called *needed* prefix.

Typically, tracing is implemented by means of labeling, i.e., the objects are labeled to be traced along the execution trace. For instances, let us consider Klop labeling for a rewrite step  $t \rightarrow s$ . A symbol in  $t$  traces to a symbol in  $s$ , if and only if the label of the former is a sublabel of the label of the latter. Note that this tracing relation is independent of the chosen tracing, while it is strictly tied to the labeling strategy.

Labeling and tracing relations in term rewriting systems have been studied in [22]. In order to study the orthogonality of execution traces, [22] investigates the equivalence of labeling and tracing along with other characterizations such as permutation, standardization, and projection. As far as we know, the use of labeling and tracing for model checking and debugging purposes has not been previously discussed in the related literature.

## 5 Backward Trace Slicing for Elementary Rewrite Theories

In this section, we formalize a backward trace slicing technique for *elementary rewrite theories* that is based on a term labeling procedure that is inspired by [6]. Since equations in  $\Delta$  are treated as rewrite rules that are used to simplify terms, our formulation for the trace slicing technique is purely based on standard rewriting.

### 5.1 Labeling procedure for rewrite theories

Let us define a labeling procedure for rules similar to [6] that allows us to trace symbols involved in a rewrite step. First, we provide the notion of labeling for terms, and then we show how it can be naturally lifted to rules and rewrite steps.

Consider a set  $\mathcal{A}$  of *atomic labels*, which are denoted by Greek letters  $\alpha, \beta, \dots$ . *Composite labels* (or simply *labels*) are defined as finite sets of elements of  $\mathcal{A}$ . By abuse, we write the label  $\alpha\beta\gamma$  as a compact denotation for the set  $\{\alpha, \beta, \gamma\}$ .

A *labeling* for a term  $t \in T_{\Sigma \cup \{\square\}}(\mathcal{V})$  is a map  $L$  that assigns a label to (the symbol occurring at) each position  $w$  of  $t$ , provided that  $\text{root}(t|_w) \neq \square$ . If  $t$  is a term, then  $t^L$  denotes the labeled version of  $t$ . Note that, in the case when  $t$  is a context, occurrences of symbol  $\square$  appearing in the labeled version of  $t$  are not labeled. The *codomain* of a labeling  $L$  is denoted by  $\text{Cod}(L) = \{l \mid (w \mapsto l) \in L\}$ .

An *initial labeling* for the term  $t$  is a labeling for  $t$  that assigns distinct fresh atomic labels to each position of the term. For example, given  $t = f(g(a, a), \square)$ , then  $t^L = f^\alpha(g^\beta(a^\gamma, a^\delta), \square)$  is the labeled version of  $t$  via the initial labeling  $L = \{A \mapsto \alpha, 1 \mapsto \beta, 1.1 \mapsto \gamma, 1.2 \mapsto \delta\}$ . This notion extends to rules and rewrite steps in a natural way as shown below.

**Labeling of Rules.** The labeling of a rewriting rule is formalized as follows:

**Definition 1.** (*rule labeling*) [6] *Given a rule  $r : \lambda \rightarrow \rho$ , a labeling  $L_r$  for  $r$  is defined by means of the following procedure.*

- $r_1$ . *The redex pattern  $\lambda^\square$  is labeled by means of an initial labeling  $L$ .*
- $r_2$ . *A new label  $l$  is formed by joining all the labels that occur in the labeled redex pattern  $\lambda^\square$  (say in alphabetical order) of the rule  $r$ . Label  $l$  is then associated with each position  $w$  of the contractum pattern  $\rho^\square$ , provided that  $\text{root}(\rho|_w^\square) \neq \square$ .*

The labeled version of  $r$  w.r.t.  $L_r$  is denoted by  $r^{L_r}$ . Note that the labeling procedure shown in Definition 1 does not assign labels to variables but only to the function symbols occurring in the rule.

**Labeling of Rewrite Steps.** Before giving the definition of labeling for a rewrite step, we need to formalize the auxiliary notion of substitution labeling.

**Definition 2.** (*substitution labeling*) Let  $\sigma = \{x_1/t_1, \dots, x_n/t_n\}$  be a substitution. A labeling  $L_\sigma$  for the substitution  $\sigma$  is defined by a set of initial labelings  $L_\sigma = \{L_{x_1/t_1}, \dots, L_{x_n/t_n}\}$  such that (i) for each binding  $(x_i/t_i)$  in the substitution  $\sigma$ ,  $t_i$  is labeled using the corresponding initial labeling  $L_{x_i/t_i}$ , and (ii) the sets  $\text{Cod}(L_{x_1/t_1}), \dots, \text{Cod}(L_{x_n/t_n})$  are pairwise disjoint.

By using Definition 2, we can formulate a labeling procedure for rewrite steps as follows.

**Definition 3.** (*rewrite step labeling*) Let  $r : \lambda \rightarrow \rho$  be a rule, and  $\mu : t \xrightarrow{r, \mathcal{G}} s$  be a rewrite step using  $r$  such that  $t = C[\lambda\sigma]_q$  and  $s = C[\rho\sigma]_q$ , for a context  $C$  and position  $q$ . Let  $\sigma = \{x_1/t_1, \dots, x_n/t_n\}$ . Let  $L_r$  be a labeling for the rule  $r$ ,  $L_C$  be an initial labeling for the context  $C$ , and  $L_\sigma = \{L_{x_1/t_1}, \dots, L_{x_n/t_n}\}$  be a labeling for the substitution  $\sigma$  such that the sets  $\text{Cod}(L_C), \text{Cod}(L_r)$ , and  $\text{Cod}(\sigma)$  are pairwise disjoint, where  $\text{Cod}(\sigma) = \bigcup_{i=1}^n \text{Cod}(L_{x_i/t_i})$ .

The rewrite step labeling  $L_\mu$  for  $\mu$  is defined by successively applying the following steps:

- $s_1$ . First, positions of  $t$  or  $s$  that belong to the context  $C$  are labeled by using the initial labeling  $L_C$ .
- $s_2$ . Then positions of  $t|_q$  (resp.  $s|_q$ ) that correspond to the redex pattern (resp. contractum pattern) of the rule  $r$  rooted at the position  $q$  are labeled according to the labeling  $L_r$ .
- $s_3$ . Finally, for each term  $t_j$ ,  $j = \{1, \dots, n\}$ , which has been introduced in  $t$  or  $s$  via the binding  $x_j/t_j \in \sigma$ , with  $x_j \in \text{Var}(\lambda)$ ,  $t_j$  is labeled using the corresponding labeling  $L_{x_j/t_j} \in L_\sigma$ .

The labeled version of a rewrite step  $\mu$  w.r.t.  $L_\mu$  is denoted by  $\mu^{L_\mu}$ . Let us illustrate these definitions by means of a rather intuitive example.

*Example 3.* Consider the rule  $r : f(g(x, y), a) \rightarrow d(s(y), y)$ . The labeled version of rule  $r$  using the initial labeling  $L = \{\Lambda \mapsto \alpha, 1 \mapsto \beta, 2 \mapsto \gamma\}$  is as follows:

$$f^\alpha(g^\beta(x, y), a^\gamma) \rightarrow d^{\alpha\beta\gamma}(s^{\alpha\beta\gamma}(y), y)$$

Consider a rewrite step  $\mu : C[\lambda\sigma] \xrightarrow{r} C[\rho\sigma]$  using  $r$ , where  $C[\lambda\sigma] = d(f(g(a, h(b)), a), a)$ ,  $C[\rho\sigma] = d(d(s(h(b)), h(b)), a)$ , and  $\sigma = \{x/a, y/h(b)\}$ . Let  $L_C = \{\Lambda \mapsto \delta, 2 \mapsto \epsilon\}$ ,  $L_{x/a} = \{\Lambda \mapsto \zeta\}$ , and  $L_{y/h(b)} = \{\Lambda \mapsto \eta, 1 \mapsto \theta\}$  be the labelings for  $C$  and the bindings in  $\sigma$ , respectively. Then, the corresponding labeled rewrite step  $\mu^L$  is as follows

$$\mu^L : d^\delta(f^\alpha(g^\beta(a^\zeta, h^\eta(b^\theta)), a^\gamma), a^\epsilon) \rightarrow d^\delta(d^{\alpha\beta\gamma}(s^{\alpha\beta\gamma}(h^\eta(b^\theta)), h^\eta(b^\theta)), a^\epsilon)$$



## 5.2 Backward Tracing Relation

Given a rewrite step  $\mu : t \xrightarrow{r} s$  and the labeling process defined in the previous section, the *backward tracing relation* computes the set of positions in  $t$  that are origin for a position  $w$  in  $s$ . Formally.

**Definition 4.** (*origin positions*) Let  $\mu : t \xrightarrow{r} s$  be a rewrite step and  $L$  be a labeling for  $\mu$  where  $L_t$  (resp.  $L_s$ ) is the labeling of  $t$  (resp.  $s$ ). Given a position  $w$  of  $s$ , the set of origin positions of  $w$  in  $t$  w.r.t.  $\mu$  and  $L$  (in symbols,  $\triangleleft_\mu^L w$ ) is defined as follows:

$$\triangleleft_\mu^L w = \{v \in \text{Pos}(t) \mid \exists p \in \text{Pos}(s), (v \mapsto l_v) \in L_t, (p \mapsto l_p) \in L_s \text{ s.t. } p \leq w \text{ and } l_v \subseteq l_p\}$$

Note that Definition 4 considers all positions of  $s$  in the path from its root to  $w$  for computing the origin positions of  $w$ . Roughly speaking, a position  $v$  in  $t$  is an origin of  $w$ , if the label of the symbol that occurs in  $t^L$  at position  $v$  is contained in the label of a symbol that occurs in  $s^L$  in the path from its root to the position  $w$ .

*Example 4.* Consider again the rewrite step  $\mu^L : t^L \rightarrow s^L$  of Example 3, and let  $w$  be the position 1.2 of  $s^L$ . The set of labeled symbols occurring in  $s^L$  in the path from its root to position  $w$  is the set  $z = \{h^\eta, d^{\alpha\beta\gamma}, d^\delta\}$ . Now, the labeled symbols occurring in  $t^L$  whose label is contained in the label of one element of  $z$  is the set  $\{h^\eta, f^\alpha, g^\beta, a^\gamma, d^\delta\}$ . By Definition 4, the set of origin positions of  $w$  in  $\mu^L$  is  $\triangleleft_{\mu^L}^L w = \{1.1.2, 1, 1.1, 1.2, \Lambda\}$ .

## 5.3 The Backward Trace Slicing Algorithm

First, let us formalize the slicing criterion, which basically represents the information we want to trace back across the execution trace in order to find out the “origins” of the data we observe. Given a term  $t$ , we denote by  $\mathcal{O}_t$  the set of *observed* positions of  $t$ .

**Definition 5.** (*slicing criterion*) Given a rewrite theory  $\mathcal{R} = (\Sigma, \Delta, R)$  and an execution trace  $\mathcal{T} : s \rightarrow^* t$  in  $\mathcal{R}$ , a *slicing criterion* for  $\mathcal{T}$  is any set  $\mathcal{O}_t$  of positions of the term  $t$ .

In the following, we show how backward trace slicing can be performed by exploiting the backward tracing relation  $\triangleleft_\mu^L$  that was introduced in Definition 4. Informally, given a slicing criterion  $\mathcal{O}_{t_n}$  for  $\mathcal{T} : t_0 \rightarrow t_2 \rightarrow \dots \rightarrow t_n$ , at each rewrite step  $t_{i-1} \rightarrow t_i$ ,  $i = 1, \dots, n$ , our technique inductively computes the backward tracing relation between the relevant positions of  $t_i$  and those in  $t_{i-1}$ . The algorithm proceeds backwards, from the final term  $t_n$  to the initial term  $t_0$ , and recursively generates at step  $i$  the corresponding set of relevant positions,  $P_{t_{n-i}}$ . Finally, by means of a removal function, a simplified trace is obtained where each  $t_j$  is replaced by the corresponding *term slice* that contains only the relevant information w.r.t.  $P_{t_j}$ .

**Definition 6.** (*sequence of relevant position sets*) Let  $\mathcal{R} = (\Sigma, \Delta, R)$  be a rewrite theory, and  $\mathcal{T} : t_0 \xrightarrow{r_1} t_1 \dots \xrightarrow{r_n} t_n$  be an execution trace in  $\mathcal{R}$ . Let  $L_i$  be the labeling for the rewrite step  $t_i \rightarrow t_{i+1}$  with  $0 \leq i < n$ . The sequence of relevant position sets in  $\mathcal{T}$  w.r.t. the slicing criterion  $\mathcal{O}_{t_n}$  is defined as follows:

$$\begin{aligned} \text{relevant\_positions}(\mathcal{T}, \mathcal{O}_{t_n}) &= [P_0, \dots, P_n] \\ \text{where } \begin{cases} P_n &= \mathcal{O}_{t_n} \\ P_j &= \bigcup_{p \in P_{j+1}} \triangleleft_{(t_j \rightarrow t_{j+1})}^{L_j} p, \text{ with } 0 \leq j < n \end{cases} \end{aligned}$$

Now, it is straightforward to formalize a procedure that obtains a term slice from each term  $t$  in  $\mathcal{T}$  and the corresponding set of relevant positions of  $t$ . We introduce the fresh symbol  $\bullet \notin \Sigma$  to replace any information in the term that is not relevant, hence does not affect the observed criterion.

**Definition 7.** (*term slice*) Let  $t \in T_\Sigma$  be a term and  $P$  be a set of positions of  $t$ . A term slice of  $t$  with respect to  $P$  is defined as follows:

$$\begin{aligned} \text{slice}(t, P) &= \text{sl\_rec}(t, P, \Lambda), \text{ where} \\ \text{sl\_rec}(t, P, p) &= \begin{cases} f(\text{sl\_rec}(t_1, P, p.1), \dots, \text{sl\_rec}(t_n, P, p.n)) & \text{if } t = f(t_1, \dots, t_n) \text{ and there exists } w \text{ s.t. } (p.w) \in P \\ \bullet & \text{otherwise} \end{cases} \end{aligned}$$

In the following, we use the notation  $t^\bullet$  to denote a term slice of the term  $t$ . Roughly speaking, the symbol  $\bullet$  can be thought of as a variable, so that any term  $t' \in \tau(\Sigma)$  can be considered as a possible concretization of  $t^\bullet$  if it is an “instance” of  $[t^\bullet]$ , where  $[t^\bullet]$  is the term that is obtained by replacing all occurrences of  $\bullet$  in  $t^\bullet$  with fresh variables.

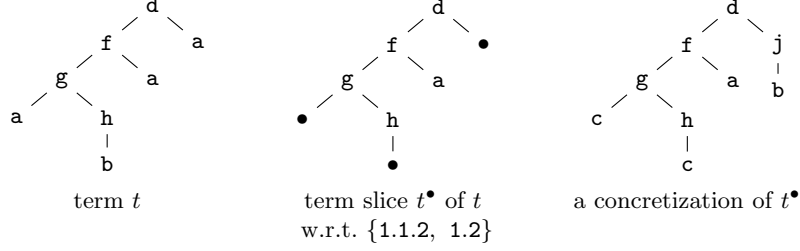
**Definition 8.** (*term slice concretization*) Given  $t' \in T_\Sigma$  and a term slice  $t^\bullet$ , we define  $t^\bullet \propto t'$  if  $[t^\bullet]$  is (syntactically) more general than  $t'$  (i.e.,  $[t^\bullet]\sigma = t'$ , for some substitution  $\sigma$ ). We also say that  $t'$  is a concretization of  $t^\bullet$ .

Figure 1 illustrates the notions of term slice and term slice concretization for a given term  $t$  w.r.t. the set of positions  $\{1.1.2, 1.2\}$ .

Let us define a *sliced rewrite step* between two term slices as follows.

**Definition 9.** (*sliced rewrite step*) Let  $\mathcal{R} = (\Sigma, \Delta, R)$  be a rewrite theory and  $r$  a rule of  $\mathcal{R}$ . The term slice  $s^\bullet$  rewrites to the term slice  $t^\bullet$  via  $r$  (in symbols,  $s^\bullet \xrightarrow{r} t^\bullet$ ) if there exist two terms  $s$  and  $t$  such that  $s^\bullet$  is a term slice of  $s$ ,  $t^\bullet$  is a term slice of  $t$ , and  $s \xrightarrow{r} t$ .

Finally, using Definition 9, backward trace slicing is formalized as follows.



**Fig. 1.** A term slice and a possible concretization.

**Definition 10.** (*backward trace slicing*) Let  $\mathcal{R} = (\Sigma, \Delta, R)$  be a rewrite theory, and  $\mathcal{T} : t_0 \xrightarrow{r_1} t_1 \dots \xrightarrow{r_n} t_n$  be an execution trace in  $\mathcal{R}$ . Let  $\mathcal{O}_{t_n}$  be a slicing criterion for  $\mathcal{T}$ , and let  $[P_0, \dots, P_n]$  be the sequence of the relevant position sets of  $\mathcal{T}$  w.r.t.  $\mathcal{O}_{t_n}$ . A trace slice  $\mathcal{T}^\bullet$  of  $\mathcal{T}$  w.r.t.  $\mathcal{O}_{t_n}$  is defined as the sliced rewrite sequence of term slices  $t_i^\bullet = \text{slice}(t_i, P_i)$  which is obtained by gluing together the sliced rewrite steps in the set

$$\mathcal{K}^\bullet = \{t_{k-1}^\bullet \xrightarrow{r_k} t_k^\bullet \mid 0 < k \leq n \wedge t_{k-1}^\bullet \neq t_k^\bullet\}.$$

Note that in Definition 10, the sliced rewrite steps that do not affect the relevant positions (i.e.,  $t_{k-1}^\bullet \xrightarrow{r_k} t_k^\bullet$  with  $t_{k-1}^\bullet = t_k^\bullet$ ) are discarded, which further reduces the size of the trace.

A desirable property of a slicing technique is to ensure that, for any concretization of the term slice  $t_0^\bullet$ , the trace slice  $\mathcal{T}^\bullet$  can be reproduced. This property ensures that the rules involved in  $\mathcal{T}^\bullet$  can be applied again to every concrete trace  $\mathcal{T}'$  that we can derive by instantiating all the variables in  $[t_0^\bullet]$  with arbitrary terms.

**Theorem 1.** (*soundness*) Let  $\mathcal{R}$  be an elementary rewrite theory. Let  $\mathcal{T}$  be an execution trace in the rewrite theory  $\mathcal{R}$ , and let  $\mathcal{O}$  be a slicing criterion for  $\mathcal{T}$ . Let  $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet \dots \xrightarrow{r_n} t_n^\bullet$  be the corresponding trace slice w.r.t.  $\mathcal{O}$ . Then, for any concretization  $t'_0$  of  $t_0^\bullet$ , it holds that  $\mathcal{T}' : t'_0 \xrightarrow{r_1} t'_1 \dots \xrightarrow{r_n} t'_n$  is an execution trace in  $\mathcal{R}$ , and  $t_i^\bullet \propto t'_i$ , for  $i = 1, \dots, n$ .

The proof of Theorem 1 relies on the fact that redex patterns are preserved by backward trace slicing. Therefore, for  $i = 1, \dots, n$ , the rule  $r_i$  can be applied to any concretization  $t'_{i-1}$  of term  $t_{i-1}^\bullet$  since the redex pattern of  $r_i$  does appear in  $t_{i-1}^\bullet$ , and hence in  $t'_{i-1}$ . A detailed proof of Theorem 1 can be found in Appendix A.

Note that our basic framework enjoys neededness of the extracted information (in the sense of [22]), since the information captured by every sliced rewrite step in a trace slice is all and only the information that is needed to produce the data of interest in the reduced term.

## 6 Backward Trace Slicing for Extended Rewrite Theories

In this section, we consider an extension of our basic slicing methodology that allows us to deal with extended rewrite theories  $\mathcal{R} = (\Sigma, E, R)$  where the equational theory  $(\Sigma, E)$  may contain associativity and commutativity axioms, and  $R$  may contain collapsing as well as nonleft-linear rules. Moreover, we also consider the built-in operators, which are not equipped with an explicit functional definition (e.g., Maude arithmetical operators). It is worth noting that all the proposed extensions are restricted to the labeling procedure of Section 5.1, keeping the backbone of our slicing technique unchanged.

### 6.1 Dealing with collapsing and nonleft-linear rules

**Collapsing Rules.** The main difficulty with collapsing rules is that they have a trivial contractum pattern, which consists in the empty context  $\square$ ; hence, it is not possible to propagate labels from the left-hand side of the rule to its right-hand side. This makes the rule labeling procedure of Definition 1 completely unproductive for trace slicing.

In order to overcome this problem, we keep track of the labels in the left-hand side of the collapsing rule  $r$ , whenever a rewrite step involving  $r$  takes place. This amounts to extending the labeling procedure of Definition 3 as follows.

**Definition 11.** (*rewrite step labeling for collapsing rules*) Let  $\mu : t \xrightarrow{r, \sigma} s$  be a rewrite step s.t.  $\sigma = \{x_1/t_1, \dots, x_n/t_n\}$ , where  $r : \lambda \rightarrow x_i$  is a collapsing rule. Let  $L_r$  be a labeling for the rule  $r$ . In order to label the step  $\mu$ , we extend the labeling procedure formalized in Definition 3 as follows:

- $s_4$ . Let  $t_i$  be the term introduced in  $s$  via the binding  $x_i/t_i \in \sigma$ , for some  $i \in \{1, \dots, n\}$ . Then, the label  $l_i$  of the root symbol of  $t_i$  in  $s$  is replaced by a new composite label  $l_c l_i$ , where  $l_c$  is formed by joining all the labels appearing in the redex pattern of  $r^{L_r}$ .

**Nonleft-linear Rules.** The trace slicing technique we described so far does not work for nonleft-linear TRS. Consider the rule:  $r : f(x, y, x) \rightarrow g(x, y)$  and the one-step trace  $\mathcal{T} : f(a, b, a) \rightarrow g(a, b)$ . If we are interested in tracing back the symbol  $g$  that occurs in the final state  $g(a, b)$ , we would get the following trace slice  $\mathcal{T}^\bullet : f(\bullet, \bullet, \bullet) \rightarrow g(\bullet, \bullet)$ . However,  $f(a, b, b)$  is a concretization of  $f(\bullet, \bullet, \bullet)$  that cannot be rewritten by using  $r$ . In the following, we augment Definition 11 in order to also deal with nonleft-linear rules.

**Definition 12.** (*rewrite step labeling for nonleft-linear rules*) Let  $\mu : t \xrightarrow{r, \sigma} s$  be a rewrite step s.t.  $\sigma = \{x_1/t_1, \dots, x_n/t_n\}$ , where  $r$  is a nonleft-linear rule. Let  $L_\sigma = \{L_{x_1/t_1}, \dots, L_{x_n/t_n}\}$  be a labeling for the substitution  $\sigma$ . In order to label the step  $\mu$ , we further extend the labeling procedure formalized in Definition 11 as follows:

- $s_5$ . For each variable  $x_j$  that occurs more than once in the left-hand side of the rule  $r$ , the following steps must be followed:

- we form a new label  $l_{x_j}$  by joining all the labels in  $\text{Cod}(L_{x_j/t})$  where  $L_{x_j/t} \in L_\sigma$ ;
- let  $l_s$  be the label of the root symbol of  $s$ . Then,  $l_s$  is replaced by a new composite label  $l_{x_j}l_s$ .

Note that, whenever a rewrite step  $\mu$  involves the application of a rule that is both collapsing and non left-linear, the labeling for  $\mu$  is obtained by sequentially applying step  $s_4$  of Definition 11 and step  $s_5$  of Definition 12 (over the labeled rewrite step resulting from  $s_4$ ).

*Example 5.* Consider the labeled, collapsing and nonleft-linear rule  $f^\beta(x, y, x) \rightarrow y$  together with the rewrite step  $\mu : h(f(a, b, a), b) \rightarrow h(b, b)$ , and matching substitution  $\sigma = \{x/a, y/b\}$ . Let  $L_{h(\square, b)} = \{\Lambda \mapsto \alpha, 2 \mapsto \epsilon\}$  be the labeling for the context  $h(\square, b)$ . Then, for the labeling  $L_\sigma = \{L_{x/a}, L_{y/b}\}$ , with  $L_{x/a} = \{\Lambda \mapsto \gamma\}$  and  $L_{y/b} = \{\Lambda \mapsto \delta\}$ , the labeled version of  $\mu$  is  $h^\alpha(f^\beta(a^\gamma, b^\delta, a^\gamma), b^\epsilon) \rightarrow h^\alpha(b^{\beta\gamma\delta}, b^\epsilon)$ . Finally, by considering the criterion  $\{1\}$ , we can safely trace back the symbol  $b$  of the sliced final state  $h(b, \bullet)$  and obtain the following trace slice

$$h(f(g(a), b, g(a)), \bullet) \rightarrow h(b, \bullet).$$

## 6.2 Built-in Operators

In practical implementations of RWL (e.g., Maude [10]), several commonly used operators are pre-defined (e.g., arithmetic operators, if-then-else constructs), which do not have an explicit specification. To overcome this limitation, we further extend our labeling process in order to deal with built-in operators.

**Definition 13.** (*rewrite step labeling for built-in operators*) For the case of a rewrite step  $\mu : C[op(t_1, \dots, t_n)] \rightarrow C[t']$  involving a call to a built-in,  $n$ -ary operator  $op$ , we extend Definition 12 by introducing the following additional case:

- $s_6$ . Given an initial labeling  $L_{op}$  for the term  $op(t_1, \dots, t_n)$ ,
- each symbol occurrence in  $t'$  is labeled with a new label that is formed by joining the labels of all the (labeled) arguments  $t_1, \dots, t_n$  of  $op$ ;
  - the remaining symbol occurrences of  $C[t']$  that are not considered in the previous step inherit all the labels appearing in  $C[op(t_1, \dots, t_n)]$ .

For example, by applying Definition 13, the addition of two natural numbers implemented through the built-in operator  $+$  might be labeled as  $+\alpha(7^\beta, 8^\gamma) \rightarrow 15^{\beta\gamma}$ .

## 6.3 Associative-Commutative Axioms

Let us finally consider an extended rewrite theory  $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ , where  $B$  is a set of associativity (A) and commutativity (C) axioms that hold for some function symbols in  $\Sigma$ . Now, since  $B$  only contains associativity/commutativity

(AC) axioms, terms can be represented by means of a single representative of their AC congruence class, called *AC canonical form* [12]. This representative is obtained by replacing nested occurrences of the same AC operator by a flattened argument list under a variadic symbol, whose elements are sorted by means of some linear ordering<sup>5</sup>. The inverse process to the flat transformation is the unflat transformation, which is nondeterministic (in the sense that it generates all the unflattened terms that are equivalent (modulo AC) to the flattened term)<sup>6</sup>.

For example, consider a binary AC operator  $f$  together with the standard lexicographic ordering over symbols. Given the  $B$ -equivalence  $f(b, f(f(b, a), c)) =_B f(f(b, c), f(a, b))$ , we can represent it by using the “internal sequence”  $f(b, f(f(b, a), c)) \rightarrow_{\text{flat}_B}^* f(a, b, b, c) \rightarrow_{\text{unflat}_B}^* f(f(b, c), f(a, b))$ , where the first one corresponds to the *flattening* transformation sequence that obtains the AC canonical form, while the second one corresponds to the inverse, unflattening one.

The key idea for extending our labeling procedure in order to cope with  $B$ -equivalence  $=_B$  is to exploit the flat/unflat transformations mentioned above. Without loss of generality, we assume that flat/unflat transformations are stable w.r.t. the lexicographic ordering over positions  $\sqsubseteq$ <sup>7</sup>. This assumption allows us to trace back arguments of commutative operators, since multiple occurrences of the same symbol can be precisely identified.

**Definition 14.** (*AC Labeling.*) Let  $f$  be an associative-commutative operator and  $B$  be the AC axioms for  $f$ . Consider the  $B$ -equivalence  $t_1 =_B t_2$  and the corresponding (internal) flat/unflat transformation  $\mathcal{T} : t_1 \rightarrow_{\text{flat}_B}^* s \rightarrow_{\text{unflat}_B}^* t_2$ . Let  $L$  be an initial labeling for  $t_1$ . The labeling procedure for  $t_1 =_B t_2$  is as follows.

1. (*flattening*) For each flattening transformation step  $t|_v \rightarrow_{\text{flat}_B} t'|_v$  in  $\mathcal{T}$  for the symbol  $f$ , a new label  $l_f$  is formed by joining all the labels attached to the symbol  $f$  in any position  $w$  of  $t^L$  s.t.  $w = v$  or  $w \geq v$ , and every symbol on the path from  $v$  to  $w$  is  $f$ ; then, label  $l_f$  is attached to the root symbol of  $t'|_v$ .
2. (*unflattening*) For each unflattening transformation step  $t|_v \rightarrow_{\text{unflat}_B} t'|_v$  in  $\mathcal{T}$  for the symbol  $f$ , the label of the symbol  $f$  in the position  $v$  of  $t^L$  is attached to the symbol  $f$  in any position  $w$  of  $t'$  such that  $w = v$  or  $w \geq v$ , and every symbol on the path from  $v$  to  $w$  is  $f$ .
3. The remaining symbol occurrences in  $t'$  that are not considered in cases 1 or 2 above inherit the label of the corresponding symbol occurrence in  $t$ .

<sup>5</sup> Specifically, Maude uses the lexicographic order of symbols.

<sup>6</sup> These two processes are typically hidden inside the  $B$ -matching algorithms that are used to implement rewriting modulo  $B$ . See [10] (Section 4.8) for an in-depth discussion on matching and simplification modulo AC in Maude.

<sup>7</sup> The lexicographic ordering  $\sqsubseteq$  is defined as follows:  $A \sqsubseteq w$  for every position  $w$ , and given the positions  $w_1 = i.w'_1$  and  $w_2 = j.w'_2$ ,  $w_1 \sqsubseteq w_2$  iff  $i < j$  or ( $i = j$  and  $w'_1 \sqsubseteq w'_2$ ). Obviously, in a practical implementation of our technique, the considered ordering among the terms should be chosen to agree with the ordering considered by flat/unflat transformations in the RWL infrastructure.

*Example 6.* Consider the transformation sequence

$$f(b, f(b, f(a, c))) \rightarrow_{\text{flat}_B}^* f(a, b, b, c) \rightarrow_{\text{unflat}_B}^* f(f(b, c), f(a, b))$$

by using Definition 14, the associated transformation sequence can be labeled as follows:

$$f^\alpha(b^\beta, f^\gamma(b^\delta, f^\epsilon(a^\zeta, c^\eta))) \rightarrow_{\text{flat}_B}^* f^{\alpha\gamma\epsilon}(a^\zeta, b^\beta, b^\delta, c^\eta) \rightarrow_{\text{unflat}_B}^* f^{\alpha\gamma\epsilon}(f^{\alpha\gamma\epsilon}(b^\beta, c^\eta), f^{\alpha\gamma\epsilon}(a^\zeta, b^\delta))$$

Note that the original order between the two occurrences of the constant  $b$  is not changed by the flat/unflat transformations. For example, in the first term,  $b^\beta$  is in position 1 and  $b^\delta$  is in position 2.1 with  $1 \sqsubseteq 2.1$ , whereas, in the last term,  $b^\beta$  is in position 1.1 and  $b^\delta$  is in position 2.2 with  $1.1 \sqsubseteq 2.2$ .

Finally, note that the methodology described in this section can be easily extended to deal with other equational attributes, e.g., identity (U), by explicitly encoding the internal transformations performed via suitable rewrite rules.

#### 6.4 Extended Soundness

Soundness of the backward trace slicing algorithm for the extended rewrite theories is established by the following theorem which properly extends Theorem 1. The proof of such an extension can be found in Appendix A.

**Theorem 2.** (*extended soundness*) Let  $\mathcal{R} = (\Sigma, E, R)$  be an extended rewrite theory. Let  $\mathcal{T}$  be an execution trace in the rewrite theory  $\mathcal{R}$ , and let  $\mathcal{O}$  be a slicing criterion for  $\mathcal{T}$ . Let  $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet \dots \xrightarrow{r_n} t_n^\bullet$  be the corresponding trace slice w.r.t.  $\mathcal{O}$ . Then, for any concretization  $t'_0$  of  $t_0^\bullet$ , it holds that  $\mathcal{T}' : t'_0 \xrightarrow{r'_1} t'_1 \dots \xrightarrow{r'_n} t'_n$  is an execution trace in  $\mathcal{R}$ , and  $t'_i \propto t'_i$ , for  $i = 1, \dots, n$ .

## 7 Experimental Evaluation

We have developed a prototype implementation of our slicing methodology that is publicly available at <http://www.dsic.upv.es/~dromero/slicing.html>. The implementation is written in Maude and consists of approximately 800 lines of code. Maude is a high-performance, reflective language that supports both equational and rewriting logic programming, which is particularly suitable for developing domain-specific applications [13]. The reflection capabilities of Maude allow metalevel computations in RWL to be handled at the object-level. This facility allows us to easily manipulate computation traces of Maude itself and eliminate the irrelevant contents by implementing the backward slicing procedures that we have defined in this paper. Using reflection to implement the slicing tool has one important additional advantage, namely, the ability to quickly integrate the tool within the Maude formal tool environment [11], which is also developed using reflection.

We developed the operator `slice` that implements the slicing process. This operator is invoked as follows:

```
slice(<moduleName>, <initialState>, <endState>, <criterion>)
```

where `moduleName` is the name of the Maude module that includes the rules and the equations to be considered in the slicing process; `initialState` and `endState` are the initial state and the final state, respectively, of the execution trace; and `criterion` is the slicing criterion. The operator works as follows. First, by considering the rules and equation in `moduleName`, the instrumented execution trace stemming from the initial state that leads to the final state is computed. Then, the slicing procedure is executed with the instrumented computation trace and the slicing criterion as inputs. Finally, a pair that contains the sliced trace and the original execution trace is delivered as outcome of the process.

In order to evaluate the usefulness of our approach, we benchmarked our prototype with several examples of Maude applications, namely: *War of Souls* (WoS), a role-playing game that is modeled as a nontrivial producer/consumer application; *Fault-Tolerant Communication Protocol* (FTCP), a Maude specification that models a fault-tolerant, client-server communication protocol; and Web-TLR, a software tool designed for model-checking real-size Web applications (e.g., Web-mailers, Electronic forums), which is based on rewriting logic.

We have tested our tool on some execution traces that were generated by the Maude applications described above by imposing different slicing criteria. For each application, we considered two execution traces that were sliced using two different criteria. As for the WoS example, we have chosen criteria that allow us to backtrack both the values produced and the entities in play — e.g., the criterion  $\text{WoS}.\mathcal{T}_1.O_2$  isolates players' behaviors along the trace  $\mathcal{T}_1$ . Execution traces in the FTCP example represent client-server interactions. In this case, the chosen criteria aim at isolating a server and a client in a scenario that involves multiple servers and clients ( $\text{FTCP}.\mathcal{T}_2.O_1$ ), and tracking the response generated by a server according to a given client request ( $\text{FTCP}.\mathcal{T}_1.O_1$ ). In the last example, we have used Web-TLR to verify two LTL(R) properties of a Webmail application. The considered execution traces are much bigger for this program, and correspond to the counterexamples produced as outcome by the built-in model-checker of Web-TLR. In this case, the chosen criteria allow us to monitor the messages exchanged by the Web browsers and the Webmail server, as well as to focus our attention on the data structures of the interacting entities (e.g., browser/server sessions, server database).

Table 1 summarizes the results we achieved. For each criterion, Table 1 shows the size of the original trace and of the computed trace slice, both measures as the length of the corresponding string. The *%reduction* column shows the percentage of reduction achieved. These results are very encouraging, and show an impressive reduction rate (up to  $\sim 95\%$ ). Actually, sometimes the trace slices are small enough to be easily inspected by the user, who can restrict her attention to the part of the computation she wants to observe getting rid of those data that are useless or even noisy w.r.t. the considered slicing criterion.



Example	Example trace	Original trace size	Slicing criterion	Sliced trace size	% reduction
WoS	WoS. $\mathcal{T}_1$	776	WoS. $\mathcal{T}_1.O_1$	201	74.10%
			WoS. $\mathcal{T}_1.O_2$	138	82.22%
	WoS. $\mathcal{T}_2$	997	WoS. $\mathcal{T}_2.O_1$	404	58.48%
			WoS. $\mathcal{T}_2.O_2$	174	82.55%
FTCP	FTCP. $\mathcal{T}_1$	2445	FTCP. $\mathcal{T}_1.O_1$	895	63.39%
			FTCP. $\mathcal{T}_1.O_2$	698	71.45%
	FTCP. $\mathcal{T}_2$	2369	FTCP. $\mathcal{T}_2.O_1$	364	84.63%
			FTCP. $\mathcal{T}_2.O_2$	707	70.16%
Web-TLR	Web-TLR. $\mathcal{T}_1$	31829	Web-TLR. $\mathcal{T}_1.O_1$	1949	93.88%
			Web-TLR. $\mathcal{T}_1.O_2$	1598	94.97%
	Web-TLR. $\mathcal{T}_2$	72098	Web-TLR. $\mathcal{T}_2.O_1$	9090	87.39%
			Web-TLR. $\mathcal{T}_2.O_2$	7119	90.13%

Table 1. Summary of the reductions achieved.

## 8 Conclusion and Related Work

We have presented a backward trace-slicing technique for rewriting logic theories. The key idea consists in tracing back —through the rewrite sequence— all the relevant symbols of the final state that we are interested in. Preliminary experiments demonstrate that the system works very satisfactorily on our benchmarks —e.g., we obtained trace slices that achieved a reduction of up to almost 95% in reasonable time (max. 0.5s on a Linux box equipped with an Intel Core 2 Duo 2.26GHz and 4Gb of RAM memory).

Tracing techniques have been extensively used in functional programming for implementing debugging tools [9]. For instance, Hat [9] is an interactive debugging system that enables exploring a computation backwards, starting from the program output or an error message (with which the computation aborted). Backward tracing in Hat is carried out by navigating a redex trail (that is, a graph-like data structure that records dependencies among function calls), whereas tracing in our approach does not require the construction of any auxiliary data structure.

Our backward tracing relation extends a previous tracing relation that was formalized in [6] for orthogonal TRSs. In [6], a label is formed from atomic labels by using the operations of sequence concatenation and underlining (e.g.,  $a$ ,  $b$ ,  $ab$ ,  $\underline{abcd}$ , are labels), which are used to keep track of the rule application order. Collapsing rules are simply avoided by coding them away. This is done by replacing each collapsing rule  $\lambda \rightarrow x$  with the rule  $\lambda \rightarrow \varepsilon(x)$ , where  $\varepsilon$  is a unary dummy symbol. Then, in order to lift the rewrite relation to terms containing  $\varepsilon$  occurrences, infinitely many new extra-rules are added that are built by saturating all left-hand sides with  $\varepsilon(x)$ . In contrast to [6], we use a simpler notion of labeling, where composite labels are interpreted as sets of atomic labels, and in the case of collapsing as well as nonleft-linear rules we

label the rewrite steps themselves so that we can deal with these rules in an effective way.

The work that is most closely related to ours is [14], which formalizes a notion of dynamic dependence among symbols by means of contexts and studies its application to program slicing of TRSs that may include collapsing as well as nonleft-linear rules. Both the *creating* and the *created* contexts associated with a reduction (i.e., the minimal subcontext that is needed to match the left-hand side of a rule and the minimal context that is “constructed” by the right-hand side of the rule, respectively) are tracked. Intuitively, these concepts are similar to our notions of redex and contractum patterns. The main differences with respect to our work are as follows. First, in [14] the slicing is given as a context, while we consider term slices. Second, the slice is obtained only on the first term of the sequence by the transitive and reflexive closure of the dependence relation, while we slice the whole execution trace, step by step. Obviously, their notion of slice is smaller, but we think that our approach can be more useful for trace analysis and program debugging. An extension of [6] is described in [22], which provides a generic definition of labeling that works not only for orthogonal TRSs as is the case of [6] but for the wider class of all left-linear TRSs. The nonleft-linear case is not handled by [22]. Specifically, [22] describes a methodology of static and dynamic tracing that is mainly based on the notion of *sample of a traced proof term* —i.e., a pair  $(\mu, P)$  that records a rewrite step  $\mu = s \rightarrow t$ , and a set  $P$  of reachable positions in  $t$  from a set of observed positions in  $s$ . The tracing proceeds forward, while ours employs a backward strategy that is particularly convenient for error diagnosis and program debugging. Finally, [14] and [22] apply to TRSs whereas we deal with the richer framework of RWL that considers equations and equational axioms, namely rewriting modulo equational theories.

## References

1. Alpuente, M., Ballis, D., Baggi, M., Falaschi, M.: A Fold/Unfold Transformation Framework for Rewrite Theories extended to CCT. In: Proc. 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, (PEPM 2010). pp. 43–52. ACM (2010)
2. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Model-checking Web Applications with Web-TLR. In: 8th Int’l Symposium on Automated Technology for Verification and Analysis (ATVA 2010). Lecture Notes in Computer Science, vol. 6252, pp. 341–346. Springer (2010)
3. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Backward Trace Slicing for Rewriting Logic Theories. In: The 23rd Int’l Conference on Automated Deduction (CADE 2011). LNCS/LNAI, Springer (2011), to appear.
4. Alpuente, M., Ballis, D., Romero, D.: Specification and Verification of Web Applications in Rewriting Logic. In: Formal Methods, Second World Congress (FM 2009). Lecture Notes in Computer Science, vol. 5850, pp. 790–805. Springer (2009)
5. Baggi, M., Ballis, D., Falaschi, M.: Quantitative Pathway Logic for Computational Biology. In: Proc. of 7th Int’l Conference on Computational Methods in Systems

- Biology (CMSB'09). Lecture Notes in Computer Science, vol. 5688, pp. 68–82. Springer (2009)
6. Bethke, I., Klop, J.W., de Vrijer, R.: Descendants and origins in term rewriting. *Inf. Comput.* 159(1-2), 59–124 (2000)
  7. Boudol, G.: Computational semantics of term rewriting systems, pp. 169–236. Cambridge University Press, New York, NY, USA (1986)
  8. Chen, F., Rosu, G.: Parametric trace slicing and monitoring. In: 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09). Lecture Notes in Computer Science, vol. 5505, pp. 246–261. Springer (2009)
  9. Chitil, O., Runciman, C., Wallace, M.: Freja, hat and hood - a comparative evaluation of three systems for tracing and debugging lazy functional programs. In: Implementation of Functional Languages, 12th International Workshop (IFL 2000). Lecture Notes in Computer Science, vol. 2011, pp. 176–193. Springer (2000)
  10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude: A High-Performance Logical Framework, Lecture Notes in Computer Science, vol. 4350. Springer-Verlag (2007)
  11. Clavel, M., Durán, F., Hendrix, J., Lucas, S., Meseguer, J., Ölveczky, P.C.: The Maude Formal Tool Environment. In: Algebra and Coalgebra in Computer Science (CALCO'07). Lecture Notes in Computer Science, vol. 4624, pp. 173–178. Springer (2007)
  12. Eker, S.: Associative-Commutative Rewriting on Large Terms. In: Proc. of 14th Int'l Conference, Rewriting Techniques and Applications (RTA '03). Lecture Notes in Computer Science, vol. 2706, pp. 14–29. Springer (2003)
  13. Eker, S., Meseguer, J., Sridharanarayanan, A.: The Maude LTL model checker and its implementation. In: Model Checking Software: Proc. 10 th Intl. SPIN Workshop. Lecture Notes in Computer Science, vol. 2648, pp. 230–234. Springer (2003)
  14. Field, J., Tip, F.: Dynamic dependence in term rewriting systems and its application to program slicing. In: Proc. of the 6th Int'l Symposium on Programming Language Implementation and Logic Programming (PLILP'94). pp. 415–431. Springer-Verlag, London, UK (1994)
  15. Hyland, M.: A syntactic characterization of the equality in some models for the lambda calculus. *J. London Math. Soc* 12, 361–370 (1976)
  16. Khasidashvili, Z.:  $\beta$ -reductions and  $\beta$ -developments of  $\lambda$ -terms with the least number of steps. In: Proc. International Conference on Computer Logic (COLOG'88). Lecture Notes in Computer Science, vol. 417, pp. 105–111. Springer (1990)
  17. Khasidashvili, Z.: Optimal normalization in orthogonal term rewriting systems. In: Proc. Rewriting Techniques and Applications (RTA'93). Lecture Notes in Computer Science, vol. 690, pp. 243–258. Springer (1993)
  18. Lévy, J.: An algebraic interpretation of the *lambda beta* k-calculus; and an application of a labelled *lambda*-calculus. *Theor. Comput. Sci.* 2(1), 97–114 (1976)
  19. Martí-Oliet, N., Meseguer, J.: Rewriting Logic: Roadmap and Bibliography. *Theoretical Computer Science* 285(2), 121–154 (2002)
  20. Riesco, A., Verdejo, A., Martí-Oliet, N.: Declarative Debugging of Missing Answers for Maude. In: 21st Int'l Conference on Rewriting Techniques and Applications (RTA 2010). LIPIcs, vol. 6, pp. 277–294. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)
  21. Talcott, C.: Pathway logic. *Formal Methods for Computational Systems Biology* 5016, 21–53 (2008)
  22. TeReSe (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge, UK (2003)

23. Wadsworth, C.P.: The Relation Between Computational and Denotational Properties for Scott's  $D_\infty$ -Models of the Lambda-Calculus. *SIAM J. Comput.* 5(3), 488–521 (1976)

## A Proofs of Theorems 1 and 2

### Proof of Theorem 1

We first demonstrate some auxiliary results which facilitate the proof of Theorem 1. The following auxiliary result is straightforward.

**Lemma 1.** *Let  $t^\bullet$  be a term slice, and let  $t'$  be a term such that  $t^\bullet \propto t'$ . For every position  $w \in \text{Pos}(t')$ , it holds that, either  $\text{root}(t'_{|w}) = \text{root}(t^\bullet_{|w})$ , or there exists a position  $u$  of  $t^\bullet$  such that  $u \leq w$  and  $\text{root}(t^\bullet_{|u}) = \bullet$ .*

*Proof.* Immediate by Definition 8.  $\square$

The following definitions are auxiliary. Let  $C$  be a context. We define the set of positions of  $C$  as the set  $\text{Pos}(C) = \{v \mid \text{root}(C_{|v}) \neq \square\}$ . Given a term  $t$ , by  $\text{path}_w(t)$ , we denote the set of symbols in  $t$  that occur in the path from its root to the position  $w$  of  $t$ , e.g.,  $\text{path}_{(2,1)}(f(a, g(b), c)) = \{f, g, b\}$ .

**Definition 15.** *Let  $r : \lambda \rightarrow \rho$  be a rule of  $\mathcal{R}$ . Let  $\mu : s \xrightarrow{r, \sigma} t$  be a rewrite step such that  $s = C[\lambda\sigma]$  and  $t = C[\rho\sigma]$ . Given a position  $w$ , we say that  $w$  is involved in  $\mu$ , if there exist  $w'$  and  $w''$  such that  $w = w'.w''$ ,  $C_{|w'} = \square$  and  $w'' \in \text{Pos}(\rho\sigma)$ .*

The following lemma establishes that, if a relevant position is involved in a rewrite step, then the origin position relation preserves the redex pattern of the rule.

**Lemma 2.** *Let  $r : \lambda \rightarrow \rho$  be a rule of an elementary rewrite theory  $\mathcal{R}$ . Let  $\mu : s \xrightarrow{r, \sigma} t$  be a rewrite step such that  $s = C[\lambda\sigma]$  and  $t = C[\rho\sigma]$ , where  $\sigma$  is a substitution and  $C$  is a context. Let  $L$  be a labeling for the rewrite step  $\mu$ , and  $w \in \text{Pos}(t)$ .*

1. *if  $w \in \text{Pos}(C)$ , then  $\triangleleft_\mu^L w = \{v \in \text{Pos}(C) \mid w = v.v'\}$*
2. *if  $w = w'.w''$ ,  $C_{|w'} = \square$ , and  $w'' \in \text{Pos}(\rho\sigma)$ , then  $\triangleleft_\mu^L w \supseteq \{w'.v' \in \text{Pos}(s) \mid v' \in \text{Pos}(\lambda)\}$*

*Proof.* Given the rule  $r : \lambda \rightarrow \rho$  and the labeling  $L$  for the rewrite step  $\mu : s \xrightarrow{r, \sigma} t$ , let us consider the labeled rewrite step  $\mu^L : s^L \xrightarrow{r^L, \sigma^L} t^L$ . By Definition 3, we can decompose the labeling  $L$  into three labelings  $L_C$ ,  $L_r$ , and  $L_\sigma$  that respectively label the context  $C$ , the redex and the contractum patterns appearing in  $\mu$ , and the terms in  $\mu$  introduced by the substitution  $\sigma$ . In other words, we have  $s^L = C^{L_C}[\lambda^{L_r} \sigma^{L_\sigma}]$  and  $t^L = C^{L_C}[\rho^{L_r} \sigma^{L_\sigma}]$ .

Let us prove the two claims independently.

**Claim 1.** We assume that  $w \in \text{Pos}(t)$  and  $w \in \text{Pos}(C)$ . Since the context  $C$  has the same initial labeling  $C^{L_C}$  in both  $s$  and  $t$ , and the sets  $\text{Cod}(L_C)$ ,  $\text{Cod}(L_r)$ , and  $\text{Cod}(L_\sigma)$  are pairwise disjoint, the set of origin positions  $\triangleleft_{s \rightarrow t}^L w$  in  $s$  is the set of positions lying on the path from the root position of  $s$  to  $w$ . Hence,  $\triangleleft_\mu^L w = \{v \in \text{Pos}(C) \mid w = v.v'\}$ .

**Claim 2.** We assume that  $w = w'.w''$ ,  $C|_{w'} = \square$ , and  $w'' \in \mathcal{Pos}(\rho\sigma)$ . Then, since  $r$  belongs to an elementary rewrite theory  $\mathcal{R}$ ,  $r$  is non-collapsing. This implies that there exists a labeled symbol  $f^{l'} \in \text{path}_w(t^L)$  belonging to the contractum pattern of the rule  $r$ . By Definition 1, for each labeled symbol  $g^l$  in the redex pattern of  $r$ , we have that  $l \subseteq l'$ . Now, since the redex pattern of  $r$  is embedded into  $s$  and the contractum pattern of  $r$  is embedded into  $t$ , the inclusion  $\triangleleft_\mu^L w \supseteq \{v.v' \in \mathcal{Pos}(s) \mid v' \in \mathcal{Pos}(\lambda)\}$  trivially holds by Definition 4.  $\square$

The following lemma establishes that, given the rewrite step  $\mu : t_0 \xrightarrow{r} t_1$  and a term slice  $t_0^\bullet$  of  $t_0$ , any concretization of  $t_0^\bullet$  is reduced by the rule  $r$  to the corresponding term slice concretization of  $t_1$ .

**Lemma 3.** *Let  $r : \lambda \rightarrow \rho$  be a rule of an elementary rewrite theory  $\mathcal{R}$ . Let  $\mu : t_0 \xrightarrow{r,\sigma} t_1$  be a rewrite step such that  $t_0 = C[\lambda\sigma]$  and  $t_1 = C[\rho\sigma]$ , where  $\sigma$  is a substitution and  $C$  is a context. Let  $L$  be a labeling for the rewrite step  $\mu$ , and  $[P_0, P_1]$  be the sequence of the relevant position sets for  $\mu : t_0 \xrightarrow{r,\sigma} t_1$  w.r.t. the slicing criterion  $\mathcal{O}$ . Let  $t_0^\bullet = \text{slice}(t_0, P_0)$ , and  $t_1^\bullet = \text{slice}(t_1, P_1)$ .*

1. *if  $P_1 \subseteq \mathcal{Pos}(C)$  then  $t_0^\bullet = t_1^\bullet$ .*
2. *if  $P_1 \cap \{w \mid w = v.v', C|_v = \square, \text{ and } v' \in \mathcal{Pos}(\rho\sigma)\} \neq \emptyset$ , then for any concretization  $t'_0$  of  $t_0^\bullet$ , we have that  $t'_0 \xrightarrow{r,\sigma'} t'_1$  where  $t_1^\bullet \propto t'_1$ .*

*Proof.* We proof the two claims separately.

**Claim 1.** Let  $P_1 \subseteq \mathcal{Pos}(C)$ . Then, by Lemma 2 (Claim 1), for any  $w \in P_1$ ,  $\triangleleft_\mu^L w = \{v \in \mathcal{Pos}(C) \mid w = v.v'\}$ . Additionally, by Definition 6,  $P_0 = \bigcup_{w \in P_1} (\triangleleft_\mu^L w)$ , and hence  $P_0 = \bigcup_{w \in P_1} \{v \in \mathcal{Pos}(C) \mid w = v.v'\}$ . Therefore, it holds that (i)  $P_1 \subseteq P_0 \subseteq \mathcal{Pos}(C)$ , and for any  $v \in P_0 \setminus P_1$ , there exists a position  $v'$  such that  $w = v.v'$  for some  $w \in P_1$ ; (ii) by Definition 7, the function  $\text{slice}(t, P)$  delivers a term slice  $t^\bullet$  where all the symbols of  $t$  that do not occur in the path connecting the root position of  $t$  with some position  $w \in P$  are abstracted by the  $\bullet$  symbol. Now, since  $t_0^\bullet = \text{slice}(t_0, P_0)$  and  $t_1^\bullet = \text{slice}(t_1, P_1)$ , by (i) and (ii), we can conclude that  $\lambda\sigma$  and  $\rho\sigma$  are abstracted by  $\bullet$ , and the context  $C$  is abstracted by the term slice  $C^\bullet$  in both  $t_0$  and  $t_1$ . Hence,  $t_0^\bullet = C^\bullet[\bullet] = t_1^\bullet$ .

**Claim 2.** We assume  $P_1 \cap \{w \mid w = v.v', C|_v = \square, \text{ and } v' \in \mathcal{Pos}(\rho\sigma)\} \neq \emptyset$ . Then, there exists a position  $w \in P_1$  such that  $w \in \{w \mid w = v.v', C|_v = \square, \text{ and } v' \in \mathcal{Pos}(\rho)\}$ . By Lemma 2 (Claim 2), it follows that  $\triangleleft_\mu^L w \supseteq \{v.v' \in \mathcal{Pos}(t_0) \mid v' \in \mathcal{Pos}(\lambda)\}$ . By Definition 6,  $P_0 = \bigcup_{w \in P_1} (\triangleleft_\mu^L w)$ , and hence  $P_0 \supseteq \{v.v' \in \mathcal{Pos}(t_0) \mid v' \in \mathcal{Pos}(\lambda)\}$ . Now, by Definition 7 and the fact that  $P_0 \supseteq \{v.v' \in \mathcal{Pos}(t_0) \mid v' \in \mathcal{Pos}(\lambda)\}$ , the redex pattern of the rule  $r$  is embedded into  $t_0^\bullet = \text{slice}(t_0, P_0)$ . In other words,  $t_0^\bullet = C^\bullet[\lambda\sigma^\bullet]$ , where  $C^\bullet$  is a term slice for the context  $C$ , and  $\sigma^\bullet$  represents the term slices for the terms introduced by the substitution  $\sigma$ . Thus, by Lemma 1, any concretization  $t'_0$  of  $t_0^\bullet$  has the form  $t'_0 = C'[\lambda\sigma']$ , where  $C^\bullet \propto C'$  and for each  $x/t \in \sigma'$ , there exists  $x/t^\bullet \in \sigma^\bullet$  such that  $t^\bullet \propto t$ . Note also that  $t_0^\bullet$  embeds the redex pattern  $\lambda^\square$  of  $r$ . Furthermore, since  $r$  belongs to the elementary rewrite theory  $\mathcal{R}$ ,  $r$  is left-linear. Thus, the

following rewrite step  $t'_0 \xrightarrow{r, \sigma'} t'_1$  can be executed for any substitution  $\sigma'$ . The rewrite step  $t'_0 \xrightarrow{r, \sigma'} t'_1$  can be decomposed as follows:  $t'_0 = C'[\lambda\sigma'] \xrightarrow{r, \sigma'} C'[\rho\sigma']$ , for some context  $C'$  and substitution  $\sigma'$ . Moreover, by definition of rewrite step,  $t'_1$  embeds the contractum pattern of  $r$ . Finally,  $t'_1 = C^\bullet[\rho^\bullet\sigma^\bullet]$ , and thus  $t'_1$  is a concretization of  $t_1^\bullet$ .  $\square$

The following proposition allows the soundness of our methodology to be proved for one-step traces on an elementary rewrite theory.

**Proposition 1.** *Let  $\mathcal{R}$  be an elementary rewrite theory. Let  $\mathcal{T}$  be an execution trace in  $\mathcal{R}$ , and let  $\mathcal{O}$  be a slicing criterion for  $\mathcal{T}$ . Let  $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet$  be the trace slice w.r.t.  $\mathcal{O}$  of  $\mathcal{T}$ . Then, for any concretization  $t'_0$  of  $t_0^\bullet$ , it holds that  $\mathcal{T}' : t'_0 \xrightarrow{r_1} t'_1$  is an execution trace in  $\mathcal{R}$  such that  $t_1^\bullet \propto t'_1$ .*

*Proof.* Given the trace slice  $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet$  w.r.t.  $\mathcal{O}$  of  $\mathcal{T}$ , let  $[P_0, P_1]$  be the sequence of the relevant position sets of  $\mathcal{T}$  w.r.t.  $\mathcal{O}$ . We have (i)  $t_0^\bullet = \text{slice}(s_0, P_0)$  and  $t_1^\bullet = \text{slice}(s_1, P_1)$ , where  $s_0 \xrightarrow{r_1} s_1$  is a rewrite step occurring in  $\mathcal{T}$ ; (ii)  $t_0^\bullet \neq t_1^\bullet$ . Let  $r_1$  be the rule  $\lambda \rightarrow \rho$ . The rewrite step  $s_0 \xrightarrow{r_1} s_1$  can be decomposed as follows:  $s_0 = C[\lambda\sigma] \xrightarrow{r_1} C[\rho\sigma] = s_1$ , for some context  $C$  and substitution  $\sigma$ .

Since  $\mathcal{R}$  is elementary and  $t_0^\bullet \neq t_1^\bullet$ , by Claim 1 of Lemma 3,  $P_1 \not\subseteq \text{Pos}(C)$ . Hence, there exists a position  $w \in P_1$  such that  $w = v.v'$  and  $v' \in \text{Pos}(\rho\sigma)$ . Also, because  $\mathcal{R}$  is elementary, we can apply Claim 2 of Lemma 3, and for any concretization  $t'_0$  of  $t_0^\bullet$ , we get  $t'_0 \xrightarrow{r_1} t'_1$  such that  $t'_1$  is a concretization of  $t_1^\bullet$ .  $\square$

**Theorem 1. (soundness)** *Let  $\mathcal{R}$  be an elementary rewrite theory. Let  $\mathcal{T}$  be an execution trace in  $\mathcal{R}$  and let  $\mathcal{O}$  be a slicing criterion for  $\mathcal{T}$ . Let  $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet \dots \xrightarrow{r_n} t_n^\bullet$  be the corresponding trace slice w.r.t.  $\mathcal{O}$ . Then, for any concretization  $t'_0$  of  $t_0^\bullet$ , it holds that  $\mathcal{T}' : t'_0 \xrightarrow{r_1} t'_1 \dots \xrightarrow{r_n} t'_n$  is an execution trace in  $\mathcal{R}$ , and  $t_i^\bullet \propto t'_i$ , for  $i = 1, \dots, n$ .*

*Proof.* The proof proceeds by induction on the length of the trace slice  $\mathcal{T}^\bullet$  and exploits Proposition 1 to prove the inductive case. Routine.  $\square$

## Proof of Theorem 2

In order to prove Theorem 2, we use the same proof scheme as for elementary rewrite theories, since the extended technique described in Section 6 is only concerned with suitable extensions of the labeling procedure given in Definition 3, which do not affect the overall backward trace slicing methodology.

Let us start by proving an extension of Lemma 2 (Claim 2), which holds for nonleft-linear as well as collapsing rules.

**Lemma 4.** *Let  $r : \lambda \rightarrow \rho$  be a rule that is either nonleft-linear or collapsing. Let  $\mu : s \xrightarrow{r, \sigma} t$  be a rewrite step such that  $s = C[\lambda\sigma]$  and  $t = C[\rho\sigma]$ , where  $\sigma$  is a substitution and  $C$  is a context. Let  $L$  be a labeling for the rewrite step  $\mu$ , and  $w \in \text{Pos}(t)$ . Then,*

1. if  $w \in \mathcal{Pos}(C)$ , then  $\triangleleft_\mu^L w = \{v \in \mathcal{Pos}(C) \mid w = v.v'\}$
2. if  $w = w'.w''$ ,  $C|_{w'} = \square$ , and  $w'' \in \mathcal{Pos}(\rho\sigma)$ , then  $\triangleleft_\mu^L w \supseteq \{w'.v' \in \mathcal{Pos}(s) \mid v' \in \mathcal{Pos}(\lambda)\}$

*Proof.* We prove the two claims separately.

**Claim 1.** The proof is identical to the proof of Claim 1 of Lemma 2.

**Claim 2.** To prove the lemma, we distinguish three cases.

**Case 1: Rule  $r$  is collapsing.** Given the collapsing rule  $r = \lambda \rightarrow \rho$  where  $\rho = x$  with  $x \in \text{Var}(\lambda)$ , let us consider the term  $t_i$  introduced by the substitution  $\sigma$  via the binding  $x/t_i$ , and we have  $\mu = C[\lambda\sigma] \xrightarrow{r} C[t_i]$ . Let us also consider the labeled rewrite step  $\mu^L : s^L \xrightarrow{r^{Lr} \sigma^{L\sigma}} t^L$  via the labeling  $L$ . By Definition 3, we have  $s^L = C^{LC}[\lambda^{Lr} \sigma^{L\sigma}]$  and  $t^L = C^{LC}[t_i^{L\sigma}]$ . Let  $f^{l'}$  be the labeled root symbol of  $t_i^{L\sigma}$ . By Definition 11 (Step  $s_4$ ), we have that  $l' = l_\lambda l_i$ , where  $l_\lambda$  is formed by joining all the labels appearing in the redex pattern  $\lambda^{Lr}$  and  $l_i$  is the label of the root of the labeled term  $t_i^{L\sigma}$ . This implies that, for each labeled symbol  $g^l$  in the redex pattern of  $r$ , we have that  $l \subseteq l'$ . Furthermore, by hypothesis, we have that  $w \in C[t_i]$  and  $w'' \in \mathcal{Pos}(t_i)$ . Hence, by Definition 4, the inclusion  $\triangleleft_\mu^L w \supseteq \{v.v' \in \mathcal{Pos}(s) \mid v' \in \mathcal{Pos}(\lambda)\}$  trivially holds.

**Case 2: rule  $r$  is nonleft-linear.** Given the nonleft-linear rule  $r$ , the proof is perfectly analogous to the proof of Lemma 2 since, by Definition 12 (Step  $s_5$ ), the label of each symbol in the contractum pattern of the rule  $r$  includes all the labels appearing in the redex pattern of  $r$ .

**Case 3: rule  $r$  is collapsing and nonleft-linear.** Since  $r$  is both collapsing and nonleft-linear,  $\mu$  is labelled according to Definition 11 (Step  $s_4$ ) and Definition 12 (Step  $s_5$ ). Therefore, we can prove the claim by simply combining the arguments used to prove Case 1 and Case 2.  $\square$

The following Lemma extends Lemma 3 to deal with collapsing and nonleft-linear rules.

**Lemma 5.** Let  $r : \lambda \rightarrow \rho$  be a rule which is either left-linear or collapsing. Let  $\mu : t_0 \xrightarrow{r, \sigma} t_1$  be a rewrite step such that  $t_0 = C[\lambda\sigma]$  and  $t_1 = C[\rho\sigma]$ , where  $\sigma$  is a substitution and  $C$  is a context. Let  $L$  be a labeling for the rewrite step  $\mu$ , and  $[P_0, P_1]$  be the sequence of the relevant position sets for  $\mu : t_0 \xrightarrow{r, \sigma} t_1$  w.r.t. the slicing criterion  $\mathcal{O}$ . Let  $t_0^\bullet = \text{slice}(t_0, P_0)$ , and  $t_1^\bullet = \text{slice}(t_1, P_1)$ . Then,

1. if  $P_1 \subseteq \mathcal{Pos}(C)$  then  $t_0^\bullet = t_1^\bullet$ .
2. if  $P_1 \cap \{w \mid w = v.v', C|_v = \square, \text{ and } v' \in \mathcal{Pos}(\rho\sigma)\} \neq \emptyset$ , then for any concretization  $t'_0$  of  $t_0^\bullet$ , we have that  $t'_0 \xrightarrow{r, \sigma'} t'_1$  where  $t_1^\bullet \propto t'_1$ .

*Proof.* We proof the two claims separately.

**Claim 1.** The proof is identical to the proof of Claim 1 of Lemma 3.

**Claim 2.** To prove the lemma, we distinguish three cases.



- Case 1: rule  $r$  is collapsing.** Given the collapsing rule  $r$ , the proof is perfectly analogous to the one of Lemma 3 Claim 2. By using Lemma 4 instead of Lemma 2, we are still able to prove that the redex pattern of  $r$  embedded in  $t_0$  is also embedded in  $t_0^\bullet$ , and hence for any concretization  $t'_0$  of  $t_0^\bullet$ , the rewrite step  $t'_0 \xrightarrow{r, \sigma'} t'_1$  can be proved. Finally, by using the same argument of Lemma 3 Claim 2, we conclude that  $t_1^\bullet \propto t'_1$ .
- Case 2: rule  $r$  is nonleft-linear.** Given the nonleft-linear rule  $r$ , the proof is similar to the one of Lemma 3. By exploiting Lemma 4 and Definition 12 (Step  $s_5$ ), we can show that (i) the redex pattern of  $r$  embedded in  $t_0$  is also embedded in  $t_0^\bullet$ , and (ii) for each term  $t$  introduced in  $t_0$  by a binding  $x/t \in \sigma$  such that  $x$  occurs multiple times in  $\lambda$ ,  $t$  is preserved in  $t_0^\bullet$  (i.e.,  $t$  is not abstracted by  $\bullet$  in  $t_0^\bullet$ ). By (i) and (ii), it is immediate to prove that, for any concretization  $t'_0$  of  $t_0^\bullet$ , the rewrite step  $t'_0 \xrightarrow{r, \sigma'} t'_1$  can be proved. Finally, by using the same argument of Lemma 3 Claim 2, we can show that  $t_1^\bullet \propto t'_1$ .
- Case 3: rule  $r$  is collapsing and nonleft-linear.** Firstly we observe that, as the rule  $r$  is collapsing, by Lemma 4 the redex pattern of  $r$  embedded in  $t_0$  is also embedded in  $t_0^\bullet$ , and hence for any concretization  $t'_0$  of  $t_0^\bullet$ , the redex pattern of  $r$  is embedded in  $t'_0$  as well. Secondly, since  $r$  is nonleft-linear, by Lemma 4 and Definition 12 (Step  $s_5$ ), for each term  $t$  introduced in  $t_0$  by a binding  $x/t \in \sigma$  such that  $x$  occurs multiple times in  $\lambda$ ,  $t$  is preserved in  $t_0^\bullet$ . Hence,  $t$  is also embedded in  $t'_0$ , for any concretization  $t'_0$  of  $t_0^\bullet$ . From the two facts above, it directly follows that for any  $t'_0$  such that  $t_0^\bullet \propto t'_0$ , the rewrite step  $t'_0 \xrightarrow{r, \sigma'} t'_1$  can be proved. Finally, by using the same argument of Lemma 3 Claim 2, we can show that  $t_1^\bullet \propto t'_1$ .

□

The following proposition allows us to prove the soundness of our methodology for one-step traces on an extended rewrite theory.

**Proposition 2.** *Let  $\mathcal{R}$  be an extended rewrite theory. Let  $\mathcal{T} : t_0 \xrightarrow{r_1} t_1$  be an execution trace in  $\mathcal{R}$ , and let  $\mathcal{O}$  be a slicing criterion for  $\mathcal{T}$ . Let  $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet$  be the trace slice w.r.t.  $\mathcal{O}$  of  $\mathcal{T}$ . Then, for any concretization  $t'_0$  of  $t_0^\bullet$ , it holds that  $\mathcal{T}' : t'_0 \xrightarrow{r_1} t'_1$  is an execution trace in  $\mathcal{R}$  such that  $t_1^\bullet \propto t'_1$ .*

*Proof.* Consider the rewrite step  $\mu : t_0 \xrightarrow{r_1} t_1$ . In the case when  $r_1$  is left-linear and non-collapsing (i.e., a rule belonging to an elementary rewrite theory), the proof is identical to the proof of Proposition 2. Hence w.l.o.g. we assume that  $r$  corresponds to a collapsing or nonleft-linear rule, built-in operator evaluation, or AC axiom.

**Nonleft-linear/collapsing rules.** In this case, the proof of Proposition 2 is analogous to the proof of Proposition 1, by using Lemma 5 in the place of Lemma 3.

**Built-in Operators.** Let  $t_0 = C[op(t_1, \dots, t_m)]$  and  $t_1 = C[t']$ . Hence,  $\mu : C[op(t_1, \dots, t_m)] \rightarrow C[t']$  is a rewrite step mimicking the evaluation of the built-in operator call  $op(t_1, \dots, t_m)$ . By Definition 13 and Definition 4, it is

immediate to show that  $op(t_1, \dots, t_m)$  is embedded in  $t_0^\bullet$ , and thus for any concretization  $t_0^\bullet \propto t'_0$ ,  $t'_0 \xrightarrow{r_1} t'_1$  and  $t_1^\bullet \propto t'_1$ .

**Associative-Commutative Axioms.** Flat/unflat transformations are interpreted as rewrite steps that reduce AC symbols. Let us first consider the flat transformation  $t \rightarrow_{flat_B} t'$  that reduces the AC symbol  $f$ . By Definition 14, the label of the occurrence of  $f$  in  $t'$  contains all the labels of the different occurrences of  $f$  appearing in  $t$  that have been reduced by the transformation. In other words, the label of  $f$  in  $t'$  keeps track of all the occurrences of  $f$  that have been reduced in  $t$ , and therefore the claim holds directly. The claim for unflat transformations can be proved in a similar way.  $\square$

Finally, we exploit Proposition 2 in order to prove the extended soundness of our methodology on extended rewrite theories.

**Theorem 2. (extended soundness)** *Let  $\mathcal{R} = (\Sigma, E, R)$  be an extended rewrite theory. Let  $\mathcal{T}$  be an execution trace in the rewrite theory  $\mathcal{R}$ , and let  $\mathcal{O}$  be a slicing criterion for  $\mathcal{T}$ . Let  $\mathcal{T}^\bullet : t_0^\bullet \xrightarrow{r_1} t_1^\bullet \dots \xrightarrow{r_n} t_n^\bullet$  be the corresponding trace slice w.r.t.  $\mathcal{O}$ . Then, for any concretization  $t'_0$  of  $t_0^\bullet$ , it holds that  $\mathcal{T}' : t'_0 \xrightarrow{r_1} t'_1 \dots \xrightarrow{r_n} t'_n$  is an execution trace in  $\mathcal{R}$  and  $t_i^\bullet \propto t'_i$ , for  $i = 1, \dots, n$ .*

*Proof.* The proof proceeds by induction on the length of the trace slice  $\mathcal{T}^\bullet$  and exploits Proposition 2 in order to prove the inductive case. Routine.  $\square$